

特開平8-194719

(43)公開日 平成8年(1996)7月30日

| (5)Int.Cl. ⁵ | 識別記号 | 庁内整理番号 | F I | 技術表示箇所 |
|-------------------------|------|---------|----------------|---------|
| G 0 6 F 17/30 | | 9194-5L | G 0 6 F 15/ 40 | 3 7 0 A |
| | | 9194-5L | 15/ 419 | 3 2 0 |

審査請求 未請求 請求項の数12 O L (全 17 頁)

| | | | |
|-------------|----------------|---------|---|
| (21)出願番号 | 特願平7-230843 | (71)出願人 | 000005223 富士通株式会社 神奈川県川崎市中原区上小田中4丁目1番1号 |
| (22)出願日 | 平成7年(1995)9月8日 | (72)発明者 | 難波 功 神奈川県川崎市中原区上小田中1015番地 富士通株式会社内 |
| (31)優先権主張番号 | 特願平6-281054 | (74)代理人 | 弁理士 大昔 義之 (外1名) |
| (32)優先日 | 平6(1994)11月16日 | | |
| (33)優先権主張国 | 日本 (J P) | | |

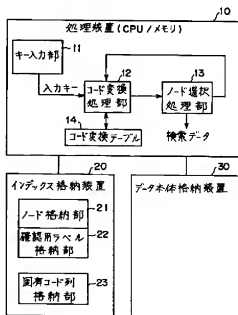
(54)【発明の名称】 検索装置および辞書/テキスト検索方法

(57)【要約】

【課題】自然言語の辞書やデータベース等のトライ法を用いる検索装置および辞書/テキスト検索方法に関し、検索の高速化とインデックスサイズの縮小化を実現することが課題である。

【解決手段】圧縮されたトライノードを格納するノード格納部21と、検索対象となる入力キーの単位コードがノードに含まれるか否かを照合するための確認用ラベルを格納する確認用ラベル格納部22と、トライ中のキー固有のコード列を格納する固有コード列格納部23とを用意し、ノード格納部21におけるノード要素を単位コードでたどりながら、確認用ラベル格納部22および固有コード列格納部23を参照して検索キーの有無を確認する。

本発明の構成例を示す図



【特許請求の範囲】

【請求項1】 キーを入力し、キーに対応するデータをトライ法により検索する検索装置において、

圧縮されたトライのノードを格納するノード格納部と、
圧縮されたトライのノードの照合の際に検索対象となる単位コードがノードに含まれるか否かを判定するためのトライのノードに対するラベルを格納するノード要素の確認用ラベル格納部と、

トライ中のキー固有のコード列を格納する固有コード列格納部と、

入力キーを1単位コードずつ取り出し、各単位コードをトライの内部コードに変換するコード変換処理部と、
単位コードがノードに含まれるか否かを前記ノード要素の確認用ラベル格納部を用いて確認し、単位コードがノードに含まれることを確認した場合に、前記ノード格納部から次のトライのノード情報または前記固有コード列格納部へのポインタ情報を得て、次の単位コードに対する照合処理に進むノード選択処理部とを備えたことを特徴とする検索装置。

【請求項2】 請求項1記載の検索装置において、
前記ノード格納部の各トライのノードには、次のトライのノードの起点となるポインタまたは前記固有コード列格納部内のキーの固有コード列へのポインタが格納され、

前記ノード選択処理部は、着目するトライのノードに次のトライのノードの起点となるポインタが格納されている場合には、その起点となるポインタと次の単位コードから得られた内部コードとによって定まるノードに前進し、着目するトライのノードに前記固有コード列格納部内のキーの固有コード列へのポインタが格納されている場合には、次の単位コード以降をポインタで示されるキーの固有コード列と照合するように構成されていることを特徴とする検索装置。

【請求項3】 請求項2記載の検索装置において、
前記ノード選択処理手段は、前記起点となるポインタの値に前記次の単位コードから得られた内部コードの値を加算して、次に着目すべきノードの位置を求め、該着目すべきノードに前記次の単位コードが対応するか否かを判定するように構成されることを特徴とする検索装置。

【請求項4】 請求項1記載の検索装置において、
入力キーとして使用される頻度の高い単位コードから順に並べたコード変換テーブルを格納するコード変換テーブル格納手段をさらに備え、

前記コード変換処理手段は、前記コード変換テーブルを用いて、使用頻度の高い単位コードほど順位の高い内部コードに変換するように構成され、前記ノード格納手段は、内部コードが表す順位に従って、対応するトライのノードを格納するように構成されることを特徴とする検索装置。

【請求項5】 請求項1記載の検索装置において、

前記確認用ラベル格納手段は、2進木を構成する複数のノード列のラベルを互いに重複しないように格納したラベル配列を格納するように構成され、前記ノード格納手段は、該ラベル配列に対応して前記次のトライの情報または前記固有コード列格納手段へのポインタ情報を格納するように構成されることを特徴とする検索装置。

【請求項6】 請求項1または請求項2記載の検索装置を用いた辞書検索方法であって、

自然言語の単語をキーとして圧縮されたトライを構成し、

自然言語の単語に対応するデータを、圧縮されたトライを用いて検索することを特徴とする辞書検索方法。

【請求項7】 請求項1または請求項2記載の検索装置を用いた辞書検索方法であって、

自然言語の音声データをキーとして圧縮されたトライを構成し、

自然言語の音声データに対応するデータを、圧縮されたトライを用いて検索することを特徴とする辞書検索方法。

【請求項8】 請求項1または請求項2記載の検索装置を用いたテキスト検索方法であって、

テキストデータをキーとして圧縮されたトライを構成し、

テキストデータに対応するデータベースのテキストを、圧縮されたトライを用いて検索することを特徴とするテキスト検索方法。

【請求項9】 キーを入力し、キーに対応するデータをトライ法により検索する検索装置における記憶媒体であって、

検索対象となる単位コードがノードに含まれるか否かを判定するためのラベルを含む圧縮されたトライのノードを作成する手段と、

トライ中のキー固有のコード列を表す固有コード列を作成する手段と、

入力キーを1単位コードずつ取り出し、各単位コードをトライの内部コードに変換する手段と、

単位コードがノードに含まれるか否かを前記ラベルを用いて確認する手段と、単位コードがノードに含まれることを確認した場合に、前記トライのノードから次のトライのノード情報と前記固有コード列へのポインタ情報のうちの1つを得て、次の単位コードに対する照合処理に進む手段とを備えることを特徴とする記憶媒体。

【請求項10】 キーを入力し、キーに対応するデータをトライ法により検索する方法において、

検索対象となる単位コードがノードに含まれるか否かを判定するためのラベルを含む圧縮されたトライのノードを作成し、

トライ中のキー固有のコード列を表す固有コード列を作成し、

入力キーを1単位コードずつ取り出し、各単位コードを

トライの内部コードに変換し、単位コードがノードに含まれるか否かを前記ラベルを用いて確認し、単位コードがノードに含まれることを確認した場合に、前記トライのノードから次のトライのノード情報と前記固有コード列へのポインタ情報のうちの1つを得て、次の単位コードに対する照合処理に進むことを特徴とする検索方法。

【請求項11】 請求項10記載の検索方法において、複数のキーの集合を表す2進木を作成し、該2進木を構成する複数のノード列のラベルを、互いに重複しないようにラベル配列に格納し、該ラベル配列に対応して、前記次のトライの情報と前記固有コード列へのポインタ情報のうちの1つをトライ配列に格納し、前記ラベル配列とトライ配列から前記圧縮されたトライのノードを作成することを特徴とする検索方法。

【請求項12】 請求項11記載の検索方法において、前記ラベル配列内の第1の位置に前記2進木の第1のノード列のラベルを格納し、前記圧縮されたトライのノード内で前記2進木の第2のノード列が挿入可能な第2の位置を求め、該第2の位置に該第2のノード列を挿入し、前記トライ配列内の前記第1の位置に、前記第2の位置を指す位置情報を格納することを特徴とする検索方法。

【発明の詳細な説明】

【0000】

【発明の属する技術分野】 本発明は検索装置および辞書／テキスト検索方法、特に高速かつ記憶容量の小さい検索装置およびそれを用いた辞書検索方法、テキスト検索方法に関する。

【0002】 機械翻訳、自然言語解析または音声認識等に用いる各種の辞書や、データベースをテキストのキーワードで検索する装置では、検索速度が高速で、かつインデックスサイズが小さいことが要求されている。

【0003】

【従来の技術】 従来の検索装置としては、ハッシュ法、B-Tree法、トライ (TRIE: tree retrieval) 法などに基づく検索方式が提案され、利用されている。

【0004】 従来のトライ法を用いた検索装置の基本概念について説明する。図21は、従来のトライ辞書検索装置の例を示している。図21において、70はトライノード選択機構、71はトライノード格納装置を表す。

【0005】 例えば英文字見出しを持つ辞書検索では、1つのトライノードは終了記号 (例えば#) を含めた (26+1) 個のトライノード要素からなり、辞書検索装置はこれらを格納するトライノード格納装置71と、トライノード選択機構70とからなる。

【0006】 図21のトライノード格納装置71では、a #, a b l e #, a g r e e # (# はキーの終了記号) をキーとするトライノードの構成例を示している。

最初のトライノードN1は、1文字目のキー情報を表しており、文字“a”が有効で、その位置に次のトライノードN2へのポインタが格納されている。2段目のトライノードN2では、2文字目の“#”と“b”と“g”の文字が有効であり、“#”の位置から検索目的のデータまたはそのデータへのアクセス情報がポイントされている。そして、トライノードN2の“b”の位置から次のトライノードN3を、“g”の位置からトライノードN6をポイントする。以下同様に、有効な文字についてのトライノードの連鎖が続く。

【0007】 トライノード選択機構70は、入力キーに対して1文字ずつ取り出し、トライノード格納装置71における最初のトライノードN1から順番に、該当する文字位置が有効なポインタ情報を保持しているかを確認し、トライノードの連鎖をたどって目的とするデータへのアクセス情報を得る。

【0008】 トライ法では、以上のようにキーの検索を先頭から1単位ずつトライノードと比較して行う。トライノード要素と文字との照合は、トライノードの文字番目の照合で行うことができるため、検索速度が高速である。

【0009】

【発明が解決しようとする課題】 従来の検索装置で用いられているハッシュ法には、キーの衝突が多く生じるような場合には制御が煩雑であり、オーバーヘッドが生じて検索速度が遅くなるか、またはキーの衝突を回避するために予め十分な大きさの記憶領域を用意しなければならないという、キーの衝突と記憶容量の問題があった。

【0010】 また、B-Tree法では、記憶容量の問題はないが、検索速度の高速性に欠けるという問題があった。一方、従来のトライ法は、検索速度は高速であるが、多くのトライノードが必要になるとともにトライノード内に無駄な領域が多いため、多大な記憶容量を必要とするという問題があり、キー数が多い場合などにはすべてのキーに対してトライ法で辞書を構成することは実際上不可能であるという問題があった。

【0011】 本発明は上記問題点の解決を図り、トライ法を用いる検索装置において検索の高速性を保つたまま、記憶容量の削減を可能とすることを目的とする。

【0012】

【課題を解決するための手段】 図1は本発明の構成例を示す。図1において、10は検索処理を実行するCPUやメモリなどからなる処理装置、11は検索キーを入力するキー入力部、12は入力キーの単位コードをトライの内部コードに変換するコード変換処理部、13はトライノードを選択して検索を進めるノード選択処理部、14はあらかじめ入力キーの単位コードとトライの内部コードとに対応情報を記憶するコード変換テーブル、20は検索キーとデータとに対応情報を管理するインデックス格納装置、21は圧縮されたトライのノードを格納す

るノード格納部、22は圧縮されたトライのノードの照合の際に検索対象となる単位コードがノードに含まれるか否かを判定するためのトライのノードに対するラベルを格納するノード要素の確認用ラベル格納部、23はトライ中のキー固有のコード列を格納する固有コード列格納部、30は検索目的のデータを格納するデータ本体格納装置を表す。

【0013】キー入力部11は、検索対象となるキーを受理し、その入力キーをコード変換処理部12に渡す。コード変換処理部12は、入力キーを1単位コードずつ取り出し、所定の計算により、またはコード変換テーブル14を用いることにより、各単位コードをトライの内部コードに変換し、ノード選択処理部13に渡す。

【0014】ノード格納部21には、従来のトライノードを重ね合わせるようにして圧縮したトライのノードが格納されている。また、確認用ラベル格納部22には、現在検索している単位コードがノード格納部21のノード上にあるかどうかの確認用ラベルが格納されている。また、固有コード列格納部23には、トライ中のキー固有のコード列が格納されている。ノード選択処理部13は、ノード格納部21および確認用ラベル格納部22を参照し、現在入力されている単位コードがトライノードにあるかどうかを判断し、単位コードがあれば次のトライノードに移り、コード変換処理部12から次の単位コードを受け取る。トライノード中に単位コードがなければ入力キーがデータ中にないと判断する。また、着目しているトライノードに固有コード列格納部23におけるキーの固有コード列へのポインタが格納されている場合には、入力キーの残りのコード列またはその内部コード列を固有コード列格納部23内のキーの固有コード列と照合する。コード列が一致してキーの照合ができれば、入力キーに対応するデータ本体格納装置30中のデータへのアクセス情報を得る。そうでなければ入力キーに対応するデータは、データ本体格納装置30中に存在しないと判断する。

【0015】本発明では、従来の個々のトライノードを有効な要素が重複しないように重ね合わせて、圧縮したトライノードを構成する。これに伴い、現在検索している単位コードがノード上にあるかどうかを確認できるように、各ノード要素に対応して確認用のラベルを格納する。また、入力キーにおけるそのキーに固有の部分は、その部分だけを取り出し、ノード格納部21とは別に設けた固有コード列格納部23に格納する。

【0016】このすることにより、検索速度は従来とほぼ同様に維持したまま記憶容量を削減することができる。図1に示す検索装置と自然言語の単語とを組み合わせることでより圧縮したトライを構成し、辞書の見出しを管理することによって、検索速度が高速でインデックスサイズが小さい仮名漢字変換辞書、形態素解析装置、構文解析装置などを実現することができる。

【0017】また、図1に示す検索装置と自然言語の音声データとを組み合わせることでより圧縮したトライを構成し、音声データに対応する認識カテゴリを管理することによって、検索速度が高速でインデックスサイズが小さい音声認識用の辞書を実現することができる。

【0018】さらにまた、図1に示す検索装置とテキストデータとを組み合わせることでより圧縮したトライを構成し、データベースのインデックスとすることによって、検索速度が高速でインデックスサイズが小さいデータベースのテキスト検索装置を実現することができる。

【0019】

【発明の実施の形態】図2は本発明の実施形態におけるコード変換テーブルとインデックス格納装置の構成例を示す図である。

【0020】例えば英文見出しを持つ辞書検索装置の場合、図2(A)に示すように、キーの終了を示す記号#と英文字a～zの文字コードとを、1～27のトライの内部コードに変換するための情報を持つコード変換テーブル14を用意する。なお、コード変換処理部12では、コード変換テーブル14を用いないで、文字コードを2進数の数値とみた算術演算により、入力キーの単位コードをトライの内部コードに変換するようにしてもよい。

【0021】図2(B)は、キーとして“a#”、“abnormal#”、“agree#”、“bachelor#”、“bcs#”の5個のキーが存在する場合のノード格納部21(TRYEと表す)と、確認用ラベル格納部22(CHECKと表す)における格納状態を示している。

【0022】“a#”、“abnormal#”、“agree#”、“bachelor#”、“bcs#”の各キーの中で他のキーとの区別に影響のない部分については、その部分を該当するキーの固有コード列として抽出し、図2(C)に示すように、固有コード列格納部23に格納する。

【0023】図2に示す辞書の作成方法について、図3に従って説明する。上述のようにキーとしては、図3(A)に示す5個のキーがあるものとする。説明を簡単にするために、これらのキーの各文字を図3(B)に示すように木構造に展開した場合を想定する。この木構造で、各キーの枝分かれのない文字列部分を抽出することにより、図3(C)に示すような固有コード列格納部23に格納すべきキーの固有コード列が得られる。なお、固有コード列格納部23における各コード列は、入力キーの元の文字コード列であっても、それをトライの内部コードに変換した後の内部コード列であっても、どちらでもよい。

【0024】ノード格納部21と確認用ラベル格納部22は、次のように作成する。まず、図3(B)に示す木

構造における従来のトライノードに相当するトライノードN1、N2、N3の部分に着目する。ノード要素の無効な部分を“0”で表し、有効な部分をその文字記号で表すと、図3(D)に示すようなトライノードN1～N3になる。これらを有効な部分が重複しないようにずらし、1本のトライノードに重ね合わせる。これをもとに、図3(D)に示すノード格納部21を作成する。なお、このノード格納部21のトライノードにおける先頭には処理の便宜上、“1”のインデックス値を持つノード要素を付加する。

【0025】例えばノード格納部21の3番目のエントリ(最初の“a”の部分)には、次のトライノードN2に相当するノード列が、トライノードN1の4番目(ノード格納部21の5番目)のエントリから始まるので、インデックス値として“4”を格納する。また、ノード格納部21の4番目のエントリ(“b”の部分)には、次のトライノードN3に相当するノード列が、トライノードN1の6番目(ノード格納部21の7番目)のエントリから始まるので、インデックス値として“6”を格納する。

【0026】また、ノード格納部21における各ノード要素に対応する単位コードの後に続くコードが、固有コード列のみからなるとき、そのノード要素に固有コード列格納部23における固有ノード列へのポインタK1～K5を格納する。なお、ノード要素がトライノードへのポインタであるか固有コード列へのポインタであるかは、例えば先頭ビットをフラグとして用いて区別することができる。より具体的には、トライノードへのポインタとして正の値を用い、固有ノード列へのポインタとして負の値を用いることが考えられる。

【0027】確認用ラベル格納部22の各エントリには、ノード格納部21における各ノード要素に対応するキーの文字コードまたは内部コードを確認用ラベルとして格納する。ノード格納部21および確認用ラベル格納部22における無効のエントリにはその旨を示すnullコード(“0”)などを設定する。

【0028】以上のようにすることによって、図2(B)、(C)に示すようなTRUE、CHECKおよび固有コード列を持つインデックス格納装置20を作成することができる。なお、ここではTRUEは、圧縮したトライのノードを格納した整数配列であり、CHECKは、文字がトライのノードに含まれるか否かを確認するための文字ラベルを格納した要素確認用の各1バイトの文字配列である。

【0029】従来のトライで構成すると、トライノードの領域として128×22要素の領域が必要であるのに対し、本実施形態では、図2から明らかなように12要素+12文字+22文字分の領域で済んでいる。

【0030】図4は、本発明の実施形態による検索処理のフローチャートである。まず、ステップS1では、入

力装置または検索要求プログラム等から検索のキーを入力する。ステップS2では、トライ検索のためのポインタをノード格納部21および確認用ラベル格納部22における最初のノードにセットし、以下、ステップS3以降を実行する。

【0031】ステップS3では、入力キーを比較するために先頭から1単位コードずつ要素を取り出す。例えば入力キーが文字列で構成される場合には、文字コードを1文字ずつ取り出すことになる。ステップS4では、取り出した単位コードをコード変換テーブル14を用いて、または算術演算もしくはビット操作により、トライの内部コードに変換する。

【0032】次にステップS5により、トライ検索のためのポインタを内部コード分進める。そして、ステップS6により、確認用ラベル格納部22を参照し、ポインタの指す確認用のラベルが現在のコードと一致するかどうかを判定する。一致しない場合には、ステップS9により検索失敗として検索要求元へ通知する。

【0033】確認用のラベルが現在のコードと等しい場合、ステップS7によってノード格納部21におけるポインタの示す要素が辞書中のキーの固有コード列格納部23を指しているかどうかを判定する。固有コード列格納部23を指していない場合、ステップS8によって、ノード格納部21のノード要素から得た値をポインタの値として、次のトライのノードにポインタを進める。その後、ステップS3に戻り、次の単位コードについて同様に処理を繰り返す。

【0034】ステップS7において、ポインタの先が辞書中の固有コード列格納部23を指しているか判定された場合には、ステップS10へ進み、キーの残りとポインタされた固有コード列とを照合する。キーの残りも固有コード列と等しい場合、ステップS11によって検索成功とし、入力キーに対するデータへのアクセス情報を得て、検索要求元へ通知する。一方、キーの残りも固有コード列と等しくない場合、ステップS12により検索失敗として検索要求元へ通知する。

【0035】次に、図4に示す処理の流れに従って、図2に示すインデックス格納装置を用いた場合の具体的な検索例について説明する。

(キー“a#”の検索要求があった場合(検索の成功例))

- (1) キー“a#”を入力する(図4のS1)。
- (2) ポインタの値を最初のノードを示すように“1”とする(S2)。
- (3) 入力キー“a#”から文字コード“a”を取り出す(S3)。
- (4) 文字コード“a”をトライの内部コード“2”に変換する(S4)。
- (5) ポインタ“1”に内部コード“2”を加算し、ポインタの値を“3”に進める(S5)。

(6) ポインタの値が“3”であることから、CHECKの第3要素に格納されている確認用ラベル“a”と現在のコード“a”とが等しいかどうかを判定する(S6)。

(7) ここでは等しいので、次にTRIEの第3要素が固有コード列を指しているかどうかを判定する(S7)。

(8) TRIEの第3要素は“4”であり、固有コード列を指していないので、ポインタの値を“4”にする(S8)。

(9) 入力キー“a#”から次の文字コード“#”を取り出す(S3)。

(10) 文字コード“#”をトライの内部コード“1”に変換する(S4)。

(11) ポインタ“4”に内部コード“1”を加算し、ポインタの値を“5”に進める(S5)。

(12) ポインタの値が“5”であることから、CHECKの第5要素に格納された確認用ラベル“#”と現在のコード“#”とが等しいかどうかを判定する(S6)。

(13) ここでは等しいので、次にTRIEの第5要素が固有コード列を指しているかどうかを判定する(S7)。

(14) TRIEの第5要素は固有コード列格納部23の“K1”を指しているため、キーの残りと“K1”の固有コード列の内容とが等しいかを判定する(S10)。

(15) キーの残りはなく、また“K1”の固有コード列の内容も空(φ)であるので、等しいと判定され、検索成功となる(S11)。

【0036】 キー“ac#”の検索要求があった場合(検索の失敗例)

(1) キー“ac#”を入力する(図4のS1)。

(2) ポインタの値を最初のノードを示すように“1”とする(S2)。

【0037】 以下、(3)から(8)までの処理は上述のキー“a#”の検索と同様である。

(9) 入力キー“ac#”から次の文字コード“c”を取り出す(S3)。

(10) 文字コード“c”をトライの内部コード“4”に変換する(S4)。

(11) ポインタ“4”に内部コード“4”を加算し、ポインタの値を“8”に進める(S5)。

(12) ポインタの値が“8”であることから、CHECKの第8要素に格納された確認用ラベル“a”と現在のコード“c”とが等しいかどうかを判定する(S6)。

(13) 確認用ラベル“a”と現在のコード“c”とは等しくないため、キー“ac#”はないことが分り、検索失敗となる(S9)。

【0038】 図5は、本発明の実施形態による日本語の単語をキーとする辞書検索装置のコード変換テーブルとインデックス格納装置の構成例を示す図である。図5の例では、“あ#”、“あさがお#”、“いろ#”、“居る#”、“居住#”、“嘘#”、“嘘つき#”の日本語文字列をキーとする辞書を構成している。

【0039】 コード変換テーブル14は、図5(A)に示すように構成され、ノード格納部21および確認用ラベル格納部22は、図5(B)に示すように構成され、固有コード列格納部23は、図5(C)に示すように構成される。これを従来のトライで構成すると約6000×13要素の領域が必要であるのに対し、本実施形態では15要素+15文字+11文字分の領域で済んでいる。

【0040】 図6は、図5に示す辞書の作成方法を示している。図6(A)は、上述の7つの日本語のキーを示している。これらのキーは、図7(B)に示すような木構造に展開される。図7(B)の木構造において、トライノードN1、N2、N3、N4に含まれない文字列部分を抽出することにより、図7(C)に示すような固有コード列が得られる。

【0041】 以下に、この辞書の検索例を説明する。
[キー“あさがお#”の検索要求があった場合(検索の成功例)]

(1) キー“あさがお#”を入力する(図4のS1)。

(2) ポインタの値を最初のノードを示すように“1”とする(S2)。

(3) 入力キー“あさがお#”から文字コード“あ”を取り出す(S3)。

(4) 文字コード“あ”をトライの内部コード“2”に変換する(S4)。

(5) ポインタ“1”に内部コード“2”を加算し、ポインタの値を“3”に進める(S5)。

(6) ポインタの値が“3”であることから、CHECKの第3要素に格納された確認用ラベル“あ”と現在のコード“あ”とが等しいかどうかを判定する(S6)。

(7) ここでは等しいので、次にTRIEの第3要素が固有コード列を指しているかどうかを判定する(S7)。

(8) TRIEの第3要素は“4”であり、固有コード列を指していないので、ポインタの値を“4”にする(S8)。

(9) 入力キーの“あさがお#”から次の文字コード“さ”を取り出す(S3)。

(10) 文字コード“さ”をトライの内部コード“4”に変換する(S4)。

(11) ポインタ“4”に内部コード“4”を加算し、ポインタの値を“8”に進める(S5)。

(12) ポインタの値が“8”であることから、CHECKの第8要素に格納された確認用ラベル“さ”と現在

のコード“さ”とが等しいかどうかを判定する（S 6）。

（13）ここでは等しいので、次にTRIEの第8要素が固有コード列を指しているかどうかを判定する（S 7）。

（14）TRIEの第8要素は固有コード列格納部23の“K3”を指しているため、キーの残りとして“K3”の固有コード列の内容とが等しいかを判定する（S 10）。

（15）キーの残りは“がお井”であり、また“K3”の位置の固有コード列も“がお井”であるため、等しいと判定され、検索成功となる（S 11）。

【0042】キー“あさ井”の検索要求があった場合（検索の失敗例）

（1）キー“あさ井”を入力する（図4のS1）。

（2）ポインタの値を最初のノードを示すように“1”とする（S2）。

【0043】以下、（3）から（13）までの処理は上述のキー“あさ井”の検索と同様である。

（14）TRIEの第8要素は固有コード列格納部23の“K3”を指しているため、キーの残りとその固有コード列“K3”とが等しいかを判定する（S 10）。

（15）キーの残りは“井”であり、また“K3”の位置の固有コード列は“がお井”であるため、等しくないとして判定され、検索失敗となる（S 12）。

【0044】次に、本発明の圧縮されたトライの作成方法について詳細に説明する。図7は、トライノードを圧縮して配列TRIEとCHECKを生成するトライ圧縮処理のフローを示している。

【0045】まず、ステップS21では、処理装置10はトライの作成対象となるキー集合をソートする。次に、ステップS22で、キー集合を構成する文字をカウントし、各文字に対して頻度の大きいものから順に、1, 2, 3, . . . , nという数字を内部コードとして割り振り、コード変換テーブル14を作成する。そして、ステップS23で、ソートされたキー集合の2進木を作成する。このとき、最小接点部分から構成され、圧縮されたトライのノードへのポインタを持った2進木を作成する。

【0046】図3（A）に示すようなキー集合からは、例えば図8のような2進木が作成される。図8の最小接点・データ領域には、図3（C）の固有コード列に対応する最小接点部またはデータが格納されている。また、このときのコード変換テーブル14は図2（A）のようになる。

【0047】次に、ステップS24以降の処理により、2進木を圧縮されたトライへ変換する。ステップS24では、配列TRIEとCHECKを初期化し、ステップS25で、2進木内の位置を示すポインタの初期状態として、それを2進木のルートノードにセットする。ま

た、ステップS26で、2進木のルートノードが挿入される配列内の位置を、index=1の位置に設定する。

【0048】ここでは、配列TRIEとCHECKの初期状態は図9のようになる。次に、ステップS27で、現在のポインタが指す2進木のノード（ポインタノード）を先頭とするノード列（ノードリスト）が挿入可能な、TRIEとCHECK内の位置を求める。挿入しようとしているノードリスト中の各要素の位置に対応する配列の領域が使用されていなければ、そのときのノードリストの先頭に対応する位置が挿入可能位置となる。この挿入可能位置は、配列の先頭から最後尾に向かって順に捜していき、最初に見つかった位置とする。

【0049】次に、ステップS28で、TRIE内の2進木挿入位置に、ステップS27で求めた挿入可能位置のindex値を設定する。例えば、2進木のポインタノードが図8のルートノードを指す場合は、ルートノードが挿入される位置はステップS26でindex=1に決まっている。このときステップS28で、index=1の位置に、挿入するリストの開始位置の値を入れる必要があるが、ルートノードの場合には初期化処理により既に設定されているため、S28の処理は省略される。

【0050】次に、ステップS29で、2進木のポインタノードより始まるリストに連なる各ノードのラベルを、CHECK内の対応する部分に書き込む。ここでは、リストに繋がっているラベルはaとbであり、それらのトライの内部コードはそれぞれ2と3である。また、ノードの挿入可能位置はindex=1の位置であるため、aの書き込み位置は1+2=3となり、bの書き込み位置は1+3=4となる。したがって、ステップS29の処理の結果は図10のようになる。

【0051】また、ステップS30で、2進木のポインタノードより始まるリストに連なる各ノードに、ステップS29でセットしたCHECKに対応するTRIEへのポインタをセットする。そして、これらのポインタが指すTRIE内の位置を2進木挿入位置とする。

【0052】ここでは、ラベルa、bに対応して、2進木のノードa、bからTRIEへポインタが張られる。これらをそれぞれ@1a、@1bと表すと、ステップS30の処理の結果は図11のようになる。

【0053】次に、ステップS31で、2進木のポインタノードがデータを持っているかどうかを確認する。それがデータを持っていないければ、ステップS32で、2進木のポインタに現在のノードの子ノードをセットして、ステップS27以降の処理を再帰的に繰り返す。

【0054】ステップS31の条件判定で、2進木のポインタノード（a→b→NULLのうちのaのノード）はデータを持っていないので、処理はステップS32に移る。ステップS32では、ポインタノードの子ノード

を2進木のポインタノードとしてセットする。この結果、ポインタノードは $\# \rightarrow b \rightarrow g \rightarrow \text{NULL}$ のうちの $\#$ のノードとなる。このステップS 32からの処理は再帰処理となり、その終了後はステップS 34から処理が始まる。

【0055】ステップS 27では、2進木の挿入可能位置を次のようにして求める。ポインタノードより始まる2進木のリスト $\# \rightarrow b \rightarrow g \rightarrow \text{NULL}$ の各ノードのラベルを、図2 (A) のコード変換テーブル14を用いてトライの内部コードに対応させると、図12のようになる。求める挿入可能位置は、上のラベルがCHECK内のラベルと重複せず、かつ、トライのノードの先頭位置となっていない位置である。ここで、トライのノードの先頭位置とは、 $\text{index} = 1$ のようにそこから始まるノードリストが存在する位置である。

【0056】挿入可能位置を探索するには、原理的には、 index の値が1の位置から順にラベルの重複がないかどうかをチェックし、候補となる位置がノードの先頭位置となっていないかどうかをチェックし、これらの条件に合う位置の index の値を返せばよい。この方法による探索は、要素がまだ挿入されていない位置に到達した時点で必ず停止するので、その終了は保証されている。

【0057】しかし、実際には計算時間が(index サイズ×挿入するノードの数)に比例し、 index の数が大きくなるとつれ実質的な時間ではなくなる。そこで、配列が index の値の小さい方から詰められ、かつ1度挿入されたノードの移動、削除がないことを利用して、配列TRIE、CHECKの使用率をモニターしながら、探索を開始する位置を調整すればよい。言い換えれば、探索開始位置の index の値を段々大きくしていけばよい。これにより、実際の探索範囲を小さくし、実用に供することができる。

【0058】こうして、ここでは図13のように、リスト $\# \rightarrow b \rightarrow g \rightarrow \text{NULL}$ の挿入可能位置が求められる。図13より、ラベル $\#$ の直前の位置が挿入可能位置となり、その index の値は4となる。

【0059】ステップS 28では、ポインタ $@1a$ が指す2進木挿入位置へステップS 27で得られた挿入可能位置の index の値4を入れる。この結果、図14のようになる。

【0060】ステップS 29では、ステップS 27で得られた挿入可能位置から2進木のポインタノードより始まるリストのラベルをCHECKに書き込む。リストに繋がっているラベルは $\#$ 、 b 、 g であり、それらのトライの内部コードはそれぞれ1、3、8である。また、ノードの挿入可能位置は $\text{index} = 4$ であるので、ステップS 29の結果は図15のようになる。

【0061】ステップS 30では、2進木から配列CHECKに対応する配列TRIEの位置へポインタを張

る。これらを $@2\#$ 、 $@2b$ 、 $@2g$ と表すと、ステップS 30の結果は図16のようになる。

【0062】ステップS 31の条件判定で、2進木のポインタノード($\# \rightarrow b \rightarrow g \rightarrow \text{NULL}$ の $\#$ のノード)はデータを持っているので、処理はステップS 33に移る。ステップS 33では、2進木のポインタノードが指すデータへのポインタを、ステップS 30でセットしたポインタが指すTRIEの要素にセットする。

【0063】ここでは、2進木の $\#$ のノードから最小接辞部へのポインタを、対応するTRIEの要素にセットする。この場合、最小接辞部の要素は空である。最小接辞・データ領域内の要素(データもしくは最小接辞部)へのポインタを $K1$ と表すと、ステップS 33の結果は図17のようになる。図17では、 $K1 = \text{空}$ である。

【0064】次に、ステップS 34では、2進木のポインタノードを同じリスト内で1つ進め、次の要素にセットする。ここでは、ポインタノードはリスト $\# \rightarrow b \rightarrow g \rightarrow \text{NULL}$ のうちの b のノードとなる。次に、ステップS 35で、2進木のポインタノードがデータを持っているかどうかをチェックする。データがあれば処理はステップS 33に移り、なければステップS 36に移る。ステップS 36では、ポインタがNULLかどうかをチェックし、NULLであれば処理を終了し、NULLでなければ処理はステップS 37に移る。

【0065】ステップS 37では、2進木のポインタに現在のノードの子ノードをセットして、ステップS 27以降の処理を再帰的に繰り返す。ステップS 38の条件判定で、2進木のポインタノード($\# \rightarrow b \rightarrow g \rightarrow \text{NULL}$ の b のうちのノード)はデータを持っているので、処理はステップS 33に移る。以下、 g のノードまで同様の処理が続く。その結果、図18に示すように、 $K1 = \text{空}$ へのポインタ、 $K2 = \text{normal}\#$ へのポインタ、 $K3 = \text{ree}\#$ へのポインタがそれぞれTRIEに格納される。ここで、2進木ポインタがNULLとなり処理は一旦終了するが、再帰処理なので呼び出し元にもどり、ステップS 34から処理が再び始まる。この場合には、2進木のポインタノードは $a \rightarrow b \rightarrow \text{NULL}$ のうちの b のノードとなる。

【0066】その後、ステップS 35、S 36の処理が行われ、 b のノードの子ノードに対してステップS 27からの一連の再帰処理が行われる。その結果、配列TRIE、CHECKは図19のようになる。図19において、 $K1 = \text{空}$ へのポインタ、 $K2 = \text{normal}\#$ へのポインタ、 $K3 = \text{ree}\#$ へのポインタ、 $K4 = \text{che}\#$ へのポインタ、 $K5 = s\#$ へのポインタを表す。

【0067】この処理が終わった段階で、2進木のポインタノードは $a \rightarrow b \rightarrow \text{NULL}$ のNULLのノードとなり、ステップS 36でポインタが空の条件に適合するため、処理は終了する。この場合には、対応する再帰処理

の呼び出し元がないので、全処理が完了することになる。こうして生成された図19のTRIE、CHECKは、図2(B)のTRIE、CHECKと一致している。

【0068】このように、処理装置10は2進木のルートから処理を開始して、2進木のルートの圧縮トライ配列への挿入場所の確保、2進木の各ノードから圧縮トライ配列へのポインタのセットの順に処理を進める。そして、ノードにデータがなければ、現在のポインタノードの子ノードを新たにポインタノードとして、再帰的に処理を行う。それ以外の場合には、2進木の各ノードから圧縮トライ配列へのポインタに対して、データへのポインタをセットし、ポインタノードをリスト上で次のノードに進める。2進木のリストの終端ノードは空であるため、2進木のルートノードからのリストの最後の処理が終了した時点で、トライ圧縮処理は終了する。

【0069】図20は、本発明の適用例を示す図である。図20(A)の例では、自然言語の単語40を入力キーとし、本発明を用いた辞書検索装置41により、辞書42の検索を行うようにしている。例えば、ワードプロセッサの仮名漢字変換辞書、自然言語解析装置や機械翻訳装置における形態素解析または構文解析に用いる辞書の応用が可能である。

【0070】図20(B)の例では、マイクロフォンなどからの音声入力50の信号をA/D変換装置51によりアナログ/ディジタル変換し、それからデータ抽出装置52によって抽出した音声特徴パラメータなどの音声データを入力キーとして、本発明を用いた辞書検索装置53により辞書54の検索を行うようにしている。本発明の技術と音声データとを組み合わせることにより圧縮したトライを構成し、効率的な音声認識等のための辞書の検索を実現することができる。

【0071】図20(C)の例では、入力キーとしてテキスト60を入力し、本発明を用いたテキスト検索装置61によりデータベース62を検索できるようにしている。これにより、テキストをキーとする日本語文書などのデータベース62の検索を、効率的に行うことが可能になる。

【0072】

【発明の効果】以上説明したように、本発明によれば、日本語文字でも、英語文字でも、速度および記憶容量の両側面において電子化辞書の検索を効率よく行うことができるようになる。また、このような辞書システムを用いることにより、ワードプロセッサの仮名漢字変換辞書、形態素解析装置、構文解析装置などの基本機能を、より効率化することが可能となる。

【図面の簡単な説明】

【図1】本発明の構成例を示す図である。

【図2】本発明の実施形態におけるコード変換テーブルとインデックス格納装置の構成例を示す図である。

【図3】図2に示す辞書の作成方法説明図である。

【図4】本発明の実施形態による検索処理のフローチャートである。

【図5】本発明の実施形態による日本語の単語をキーとする辞書検索装置のコード変換テーブルとインデックス格納装置の構成例を示す図である。

【図6】図5に示す辞書の作成方法説明図である。

【図7】トライ圧縮処理のフローチャートである。

【図8】英語の2進木の例を示す図である。

【図9】トライノード圧縮時の配列データの例を示す図(その1)である。

【図10】トライノード圧縮時の配列データの例を示す図(その2)である。

【図11】トライノード圧縮時の配列データの例を示す図(その3)である。

【図12】ラベルから内部コードへの変換例を示す図である。

【図13】トライノード圧縮時の配列データの例を示す図(その4)である。

【図14】トライノード圧縮時の配列データの例を示す図(その5)である。

【図15】トライノード圧縮時の配列データの例を示す図(その6)である。

【図16】トライノード圧縮時の配列データの例を示す図(その7)である。

【図17】トライノード圧縮時の配列データの例を示す図(その8)である。

【図18】トライノード圧縮時の配列データの例を示す図(その9)である。

【図19】トライノード圧縮時の配列データの例を示す図(その10)である。

【図20】本発明の適用例を示す図である。

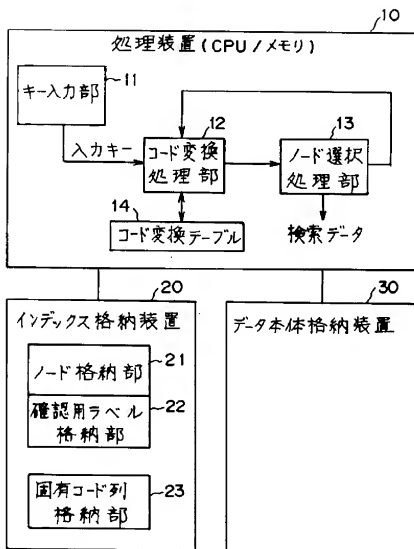
【図21】従来のトライ辞書検索装置の例を示す図である。

【符号の説明】

- 10 処理装置
- 11 キー入力部
- 12 コード変換処理部
- 13 ノード遷移処理部
- 14 コード変換テーブル
- 20 インデックス格納装置
- 21 ノード格納部
- 22 確認用ラベル格納部
- 23 固有コード列格納部
- 30 データ本体格納装置

【図1】

本発明の構成例を示す図



【図12】

ラベルから内部コードへの変換例を示す図

| # | b | g |
|---|---|---------------|
| 1 | 0 | 3 0 0 0 0 0 8 |

【図14】

トライノード圧縮時の配列データの例を示す図
(その5)

| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-------|---|---|---|---|---|---|---|---|---|----|----|----|
| TRIE | 1 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CHECK | 0 | 0 | a | b | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

【図2】

コード変換テーブルとインデックス格納装置の構成例

(A) コード変換テーブル

14

| 文字コード | # | a | b | c | d | e | f | g | ... | z |
|-----------|---|---|---|---|---|---|---|---|-----|----|
| トライの内蔵コード | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ... | 27 |

(B) 圧縮されたトライと要素確認用の配列

21
22

| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-------|---|---|---|---|----|---|----|----|---|----|----|----|
| TRIE | 1 | 0 | 4 | 6 | K1 | 0 | K2 | K4 | 0 | K5 | 0 | K3 |
| CHECK | 0 | 0 | a | b | # | 0 | b | a | 0 | c | 0 | * |

(C) キーの固有コード列

23

| | K1 | K2 | K3 | K4 | K5 |
|--------|----|---------|------|---------|----|
| 固有コード列 | # | normal# | ree# | chelor# | s# |

【図8】

【図5】

コード変換テーブルとインデックス格納装置の構成例

(A) コード変換テーブル

14

| 文字コード | # | あい | さ | る | つ | む | ... |
|-----------|---|----|---|---|---|---|-----|
| トライの内蔵コード | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

(B) 圧縮されたトライと要素確認用の配列

21
22

| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|----|----|----|----|---|----|---|----|----|----|----|----|----|
| TRIE | 1 | 0 | 4 | K1 | K2 | K3 | 0 | K4 | 5 | K7 | 0 | K4 | 0 | K5 | |
| CHECK | 0 | 0 | あい | # | # | 0 | さ | る | む | つ | 0 | る | 0 | む | |

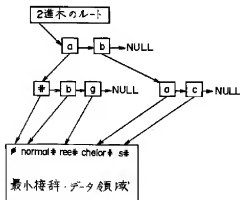
(C) キーの固有コード列

23

| | K1 | K2 | K3 | K4 | K5 | K6 | K7 |
|--------|----|----|-----|----|----|----|----|
| 固有コード列 | ろ# | # | あゆ# | # | # | # | ろ# |

【図9】

更迭の2進木の例を示す図



トライノード圧縮時の配列データの例を示す図

(その1)

| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|---|---|---|---|---|---|---|---|
| TRIE | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CHECK | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

【図10】

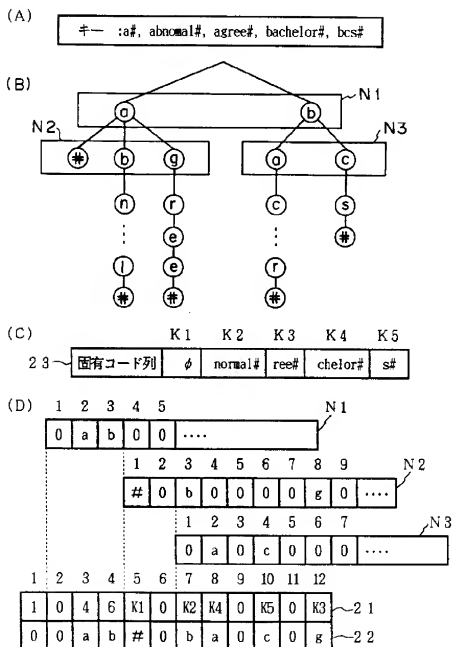
トライノード圧縮時の配列データの例を示す図

(その2)

| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------|---|---|---|---|---|---|---|---|
| TRIE | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CHECK | 0 | 0 | a | b | 0 | 0 | 0 | 0 |

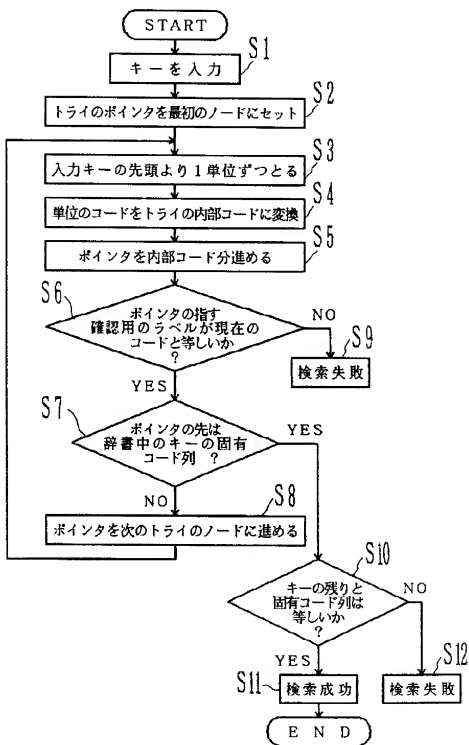
【図3】

図2に示す辞書の作成方法を説明する図



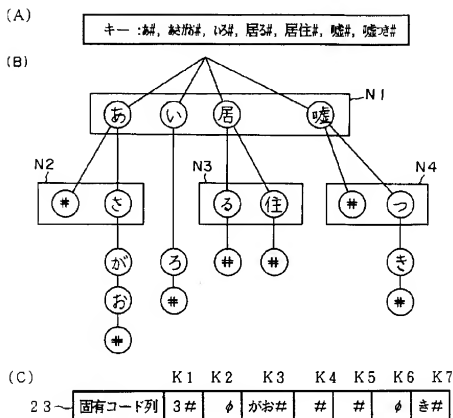
【図4】

本発明の実施形態による検索処理のフローチャート



【図6】

図5に示す辞書の作成方法説明図



【図11】

トライノード圧縮時の配列データの例を示す図
(その3)

| | | | | | | | | |
|-------|---|---|-----|-----|---|---|---|---|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| TRIE | 1 | 0 | @1a | @1b | 0 | 0 | 0 | 0 |
| CHECK | 0 | 0 | a | b | 0 | 0 | 0 | 0 |

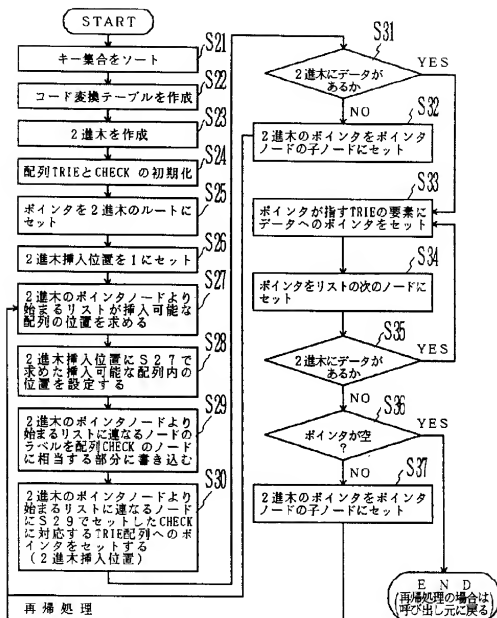
【図13】

トライノード圧縮時の配列データの例を示す図
(その4)

| | | | | | | | | | | | | |
|-------|---|---|-----|-----|---|---|---|---|---|----|----|----|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| TRIE | 1 | 0 | @1a | @1b | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CHECK | 0 | 0 | a | b | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | | | # | 0 | b | 0 | 0 | 0 | 0 | g |

【図7】

トライ圧縮処理のフローチャート



【図15】

トライノード圧縮時の配列データの例を示す図

(その6)

| | | | | | | | | | | | | |
|-------|---|---|---|----|---|---|---|---|---|----|----|----|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| TRIE | 1 | 0 | 4 | 0b | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CHECK | 0 | 0 | a | b | # | 0 | b | 0 | 0 | 0 | 0 | g |

【図17】

トライノード圧縮時の配列データの例を示す図

(その8)

| | | | | | | | | | | | | |
|-------|---|---|---|----|----|---|----|---|---|----|----|----|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| TRIE | 1 | 0 | 4 | 0b | K1 | 0 | 0b | 0 | 0 | 0 | 0 | 0b |
| CHECK | 0 | 0 | a | b | # | 0 | b | 0 | 0 | 0 | 0 | g |

【図19】

トライノード圧縮時の配列データの例を示す図

(その10)

| | | | | | | | | | | | | |
|-------|---|---|---|---|----|---|----|----|---|----|----|----|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| TRIE | 1 | 0 | 4 | 6 | K1 | 0 | K2 | K4 | 0 | K5 | 0 | K3 |
| CHECK | 0 | 0 | a | b | # | 0 | b | a | 0 | c | 0 | g |

【図16】

トライノード圧縮時の配列データの例を示す図

(その7)

| | | | | | | | | | | | | |
|-------|---|---|---|----|-----|---|----|---|---|----|----|----|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| TRIE | 1 | 0 | 4 | 0b | 02# | 0 | 0b | 0 | 0 | 0 | 0 | 0b |
| CHECK | 0 | 0 | a | b | # | 0 | b | 0 | 0 | 0 | 0 | g |

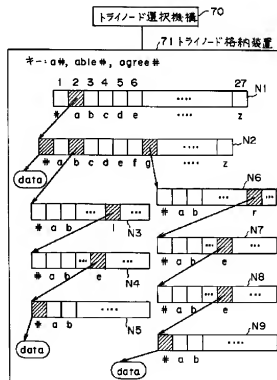
【図18】

トライノード圧縮時の配列データの例を示す図

(その9)

| | | | | | | | | | | | | |
|-------|---|---|---|----|----|---|----|---|---|----|----|----|
| INDEX | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| TRIE | 1 | 0 | 4 | 0b | K1 | 0 | K2 | 0 | 0 | 0 | 0 | K3 |
| CHECK | 0 | 0 | a | b | # | 0 | b | 0 | 0 | 0 | 0 | g |

【図21】



【図20】

本発明の適用例を示す図

